NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

Priority: IST-2004-2.4.7 — "Semantic-based knowledge and content systems"

D3.3.3: Integration of OntoLight with the Alignment server

Deliverable Co-ordinator: **Jérôme Euzenat**

Deliverable Co-ordinating Institution: **INRIA**

Other Authors: **Jérôme David (INRIA), Marko Grobelnik (JSI), Chan Le Duc (INRIA), Boštjan Pajntar (JSI), Dunja Mladenić(JSI)**

This deliverable describes the integration of the OntoLight matcher within the Alignment server and the NeOn toolkit. This integration uses a web service connection from the Alignment server to an OntoLight web service interface.

Document Identifier:	NEON/2008/D3.3.3/v1.0	Date due:	October 31st, 2008
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	October 31st, 2008
Project start date	March 1, 2006	Version:	v1.0
Project duration:	4 years	State:	Final
		Distribution:	Public

NeOn Consortium

This document is part of the NeOn research project funded by the IST Programme of the Commission of the European Communities by the grant number IST-2005-027595. The following partners are involved in the project:

Open University (OU) – Coordinator Knowledge Media Institute – KMi Berrill Building, Walton Hall Milton Keynes, MK7 6AA United Kingdom Contact person: Martin Dzbor, Enrico Motta E-mail address: {m.dzbor, e.motta}@open.ac.uk	**Universität Karlsruhe – TH (UKARL)** Institut für Angewandte Informatik und Formale Beschreibungsverfahren – AIFB Englerstrasse 11 D-76128 Karlsruhe, Germany Contact person: Peter Haase E-mail address: pha@aifb.uni-karlsruhe.de
Universidad Politécnica de Madrid (UPM) Campus de Montegancedo 28660 Boadilla del Monte Spain Contact person: Asunción Gómez Pérez E-mail address: asun@fi.ump.es	**Software AG (SAG)** Uhlandstrasse 12 64297 Darmstadt Germany Contact person: Walter Waterfeld E-mail address: walter.waterfeld@softwareag.com
Intelligent Software Components S.A. (ISOCO) Calle de Pedro de Valdivia 10 28006 Madrid Spain Contact person: Jesús Contreras E-mail address: jcontreras@isoco.com	**Institut 'Jožef Stefan' (JSI)** Jamova 39 SL–1000 Ljubljana Slovenia Contact person: Marko Grobelnik E-mail address: marko.grobelnik@ijs.si
Institut National de Recherche en Informatique et en Automatique (INRIA) ZIRST – 665 avenue de l'Europe Montbonnot Saint Martin 38334 Saint-Ismier, France Contact person: Jérôme Euzenat E-mail address: jerome.euzenat@inrialpes.fr	**University of Sheffield (USFD)** Dept. of Computer Science Regent Court 211 Portobello street S14DP Sheffield, United Kingdom Contact person: Hamish Cunningham E-mail address: hamish@dcs.shef.ac.uk
Universität Kolenz-Landau (UKO-LD) Universitätsstrasse 1 56070 Koblenz Germany Contact person: Steffen Staab E-mail address: staab@uni-koblenz.de	**Consiglio Nazionale delle Ricerche (CNR)** Institute of cognitive sciences and technologies Via S. Marino della Battaglia 44 – 00185 Roma-Lazio Italy Contact person: Aldo Gangemi E-mail address: aldo.gangemi@istc.cnr.it
Ontoprise GmbH. (ONTO) Amalienbadstr. 36 (Raumfabrik 29) 76227 Karlsruhe Germany Contact person: Jürgen Angele E-mail address: angele@ontoprise.de	**Food and Agriculture Organization of the United Nations (FAO)** Viale delle Terme di Caracalla 00100 Rome Italy Contact person: Marta Iglesias E-mail address: marta.iglesias@fao.org
Atos Origin S.A. (ATOS) Calle de Albarracín, 25 28037 Madrid Spain Contact person: Tomás Pariente Lobo E-mail address: tomas.parientelobo@atosorigin.com	**Laboratorios KIN, S.A. (KIN)** C/Ciudad de Granada, 123 08018 Barcelona Spain Contact person: Antonio López E-mail address: alopez@kin.es

Work package participants

The following partners have taken an active part in the work leading to the elaboration of this document, even if they might not have directly contributed to the writing of this document or its parts:

- Jožef Stefan Institute
- Open university
- iSoCo
- Universität Karlsruhe
- INRIA
- Ontoprise
- CNR
- Universidad Politécnica de Madrid

Change Log

Version	Date	Author	Changes
0.1	15.04.2008	Jérôme Euzenat	initial version
0.2	26.06.2008	Jérôme Euzenat	interface description after discussing with M. Grobelnik
0.3	21.08.2008	Jérôme Euzenat	updated with the actual implementated messages
0.4	27.08.2008	Jérôme Euzenat	added WSDL
0.5	20.08.2008	Jérôme Euzenat	enhanced texts
0.6	25.09.2008	Jérôme Euzenat	featured empty sections
0.7	5.10.2008	Boštjan Pajntar	Section 2.2
0.8	12.10.2008	Boštjan Pajntar	Chapter 4
0.9	17.10.2008	Dunja Mladenić	Revision of JSI sections
0.10	9.11.2008	Boštjan Pajntar	Added WSDL files in chapter 4.
0.11	20.11.2008	Marko Grobelnik	Overall revision.
0.12	5.12.2008	Dunja Mladenić	Overall revision.
1.0	20.12.2008	Jérôme Euzenat	Taken quality control into account

Executive Summary

In the NeOn project, context can be expressed within a network of ontologies through relationships between ontologies. Such relationships would provide the missing axioms that specify the meaning of knowledge further and help considering them in their context.

These relations must be established from the initial resources which will be linked to contextual resources. This can be achieved through the use of ontology matching techniques. In order to help networked ontology designers to contextualise ontologies, we have already integrated within the NeOn toolkit a plug-in for finding alignments between ontologies and importing them within the network of ontologies.

There are systems which cannot be integrated within the NeOn toolkit or even within an Alignment server either because they depend on local resources, e.g., databases, they consume large amount of resources or they cannot be easily ported. For these systems to be adequately exploited by the Alignment server and the NeOn Alignment plug-in, it is necessary to establish connection.

This deliverable presents the work done for the introduction of the OntoLight matcher within the Alignment server which can compute and provide alignments for the NeOn toolkit Alignment plug-in. More generally, it provides a web service interface specification which enables any matcher to be interfaced with the Alignment server.

For that purpose, we have defined a new kind of alignment method which connects to a web service for performing the match. The web service interface is specified in WSDL and is used for connecting the Alignment server to OntoLight.

Contents

Chapter 1

Introduction

In this chapter, we recast the intuition presented in deliverables D3.1.1 and D3.3.1 in the context of networked ontologies. In particular, we explain in what sense contexts are networks of ontologies (§1.1), what are the support developed within NeOn for supporting ontology matching and alignment sharing (§1.2) and how this integrates within the NeOn architecture (§1.3). In this last section, we introduce the need to have a new kind of connection for the Alignment server to be connected with heavy stand-alone systems like OntoLight.

1.1 Contextualising ontologies through matching

Domain ontologies are designed to be applied in a particular context and use terms in a sense that is relevant to this domain, e.g., *Ontology* in computer science, and which may not be related to similar concepts in other domains. They do not fully specify concepts because part of their specification is implicit from the context. We will use here the word “constraint” for this specification in a broad sense: any axiom constrains the meaning of the terms it uses.

NeOn has in its core the ambitious scenario that ontologies are developed in an open environment in a distributed fashion. This means that ontologies will be used in settings that are not those that have led to their design. In order to use them properly, it is necessary to be able to identify this context so that appropriate actions can be taken: either not using them or adapting them to the new context.

The context of some knowledge or ontology is given by additional knowledge in which or in the perspective of which this knowledge has been elaborated. This knowledge can vary in nature and expression. For instance, in work package 2, the context of elaboration of ontologies is expressed as argumentative structures about ontology design rationale. In work package 3 [Haase *et al.*, 2006], the context of ontologies is prominently placed in the framework of networked ontologies: the context of an ontology is given by the network of other ontologies with which it is related.

From this definition, a pair of dual operations can be associated with context (see Figure 1.1): contextualisation and decontextualisation. Contextualisation or recontextualisation is the action of finding the relations of an ontology with other ontologies which express its context. Decontextualisation, as the opposite, extracts one particular ontology from its context. These operation can be combined, for instance, if someone wants to transfer one ontology from a domain to another one, by first decontextualising it and recontextualising it to another domain.

As can be considered from this brief description, contextualising an ontology is a matter of matching it to other ontologies which will provide its context. For that purpose ontology matching technologies can be used. So we would like to provide support for contextualising ontologies through ontology matching. Moreover, recontextualising ontologies can be succesfully used for helping the process of matching an ontology into another one. Thus, we also aim at taking into account the contextualisation operation for matching.

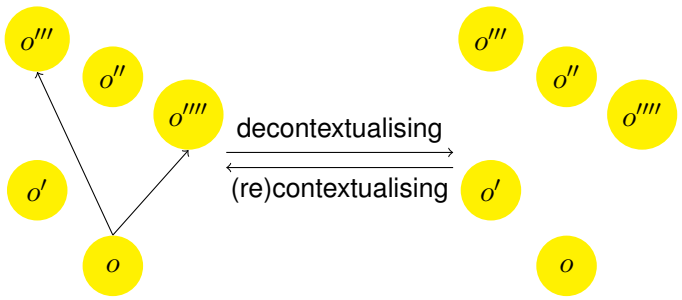

Figure 1.1: Contextualising/Decontextualising ontologies in the framework of networked ontologies.

1.2 Support for matching ontologies

In Deliverable D3.3.1, we have considered expressing context with two main constructions from the NeOn model for networked ontologies: Ontologies and mappings (or alignments). The NeOn model already offers the metamodel for these two kinds of entities. The Alignment API can be considered as an implementation of this metamodel.

The Alignment API [Euzenat, 2004] has been designed to help developing applications based on alignments. It has been developed in the aim of manipulating a standard alignment format for sharing among matching systems, but it provides the features required for sharing them more widely. The API is a JAVA description of tools for accessing alignments in the format presented above.

The Alignment API can be used in conjunction with an ontology language API (the OWL-API is currently available, but other instantiation could be based on totally different languages). This implementation offers the following services:

- Computing and representing alignments;
- Piping alignments algorithms (for improving an existing alignment);
- Manipulating (trimming and hardening) and combining (merging, composing) alignments;
- Generating “mediators” (transformations, axioms, rules in format such as XSLT, SWRL, OWL, C-OWL, WSML);
- Comparing alignments (like computing precision and recall or a symmetric distance with regard to a particular reference alignment).

The API also provides the ability to compose matching algorithms and manipulating alignments through programming. The API can be used for producing transformations, rules or bridge axioms independently from the algorithm that produced the alignment. Since its definition, several matching systems have been developed within this API (OLA, oMap) and more of them are able to generate its format (FOAM, Prompt, Falcon, etc.).

Within task 3.3 of NeOn we have developed an Alignment server (see Deliverable 3.3.1) which allows sharing alignments. The Alignment server is built around the Alignment API. The server architecture is made of three layers (shown in Figure 1.2):

A storage system that allows persistent storage and retrieval of alignments. It implements only basic storage and runtime memory caching functions. The storage is made through a DBMS interface and can be replaced by any database management system as soon as it is supported by jdbc.

A protocol manager which handles the server protocol. It accepts the queries from plug-in interfaces and uses the server resources for answering them. It uses the storage system for caching results.

Protocol plug-ins which accept incoming queries in a particular communication system and invoke the protocol manager in order to satisfy them. These plug-ins are ideally stateless and only translator for the external queries.

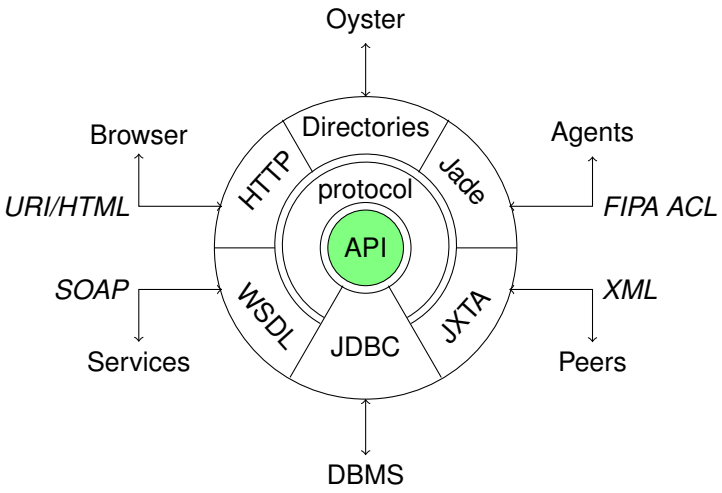

Figure 1.2: The Alignment server is built on the Alignment API that is seated on top of a relational database repository for alignment and is wrapped around a simple protocol. Each access method is a plug-in that interacts with the server through the protocol. Currently, HTML, agent and web service plug-in are available.

Currently, three plug-ins are available for the server:

- HTTP/HTML plug-in for interacting through a browser;
- JADE/FIPA ACL for interacting with agents;
- HTTP/SOAP plug-in for interacting as a web service.

Services that are provided by the Alignment server are:

- storing alignments, whether they are provided by automatic means or by hand;
- storing annotations in order for the clients to evaluate them and to decide to use one of them or to start from it (this starts with the information about the matching algorithms, the justifications for correspondences that can be used in agent argumentation, as well as properties of the alignment);
- producing alignments on the fly through various algorithms that can be extended and parametrised;
- manipulating alignments by inverting them, applying thresholds;
- generating knowledge processors such as mediators, transformations, translators, rules as well as to process these processors if necessary.

There is no constraint that the alignments are computed online or off-line (i.e., they are stored in the alignment store) or that they are processed by hand or automatically. This kind of information can however be stored together with the alignment in order for the client to be able to discriminate among them. For applications, the Alignment server can be available:

at design time through invocation by design and engineering environments: It can be integrated within the development environment.

at run time through the web service access of the server (or any other available plug-in).

1.3 The NeOn Alignment plug-in and Alignment server extensions

The NeOn Alignment plug-in, described in Deliverable 3.3.2 [Le Duc *et al.*, 2008], is a NeOn toolkit plug-in that takes advantage of both the Alignment API and Alignment servers. It thus provides design time support for alignment management from the NeOn toolkit.

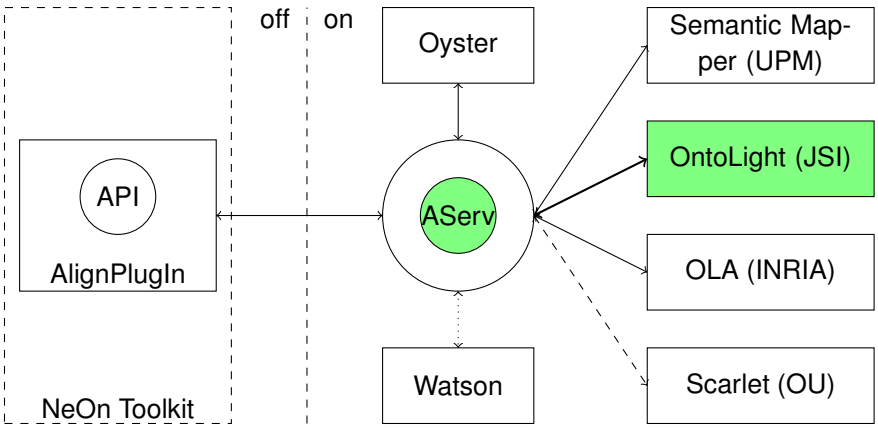

Figure 1.3: The integration of the Alignment server within the NeOn architecture (thick line corresponds to connection presented in this deliverable, regular lines to connections presented in D3.3.2 and dotted lines to future connections.

The NeOn Alignment plug-in can work in stand alone mode thanks to its embedding of the Alignment API or in connection with an Alignment server. In the latter case, it will benefit from new matchers integrated within the Alignment server and alignment sharing. The purpose of this deliverable is to describe a third way of integrating new matchers through web services and, in particular, the OntoLight matcher. Figure 1.3 presents the design-time support provided in NeOn in relation with Alignment servers and external matchers. Hence the goal of the work presented in this deliverable is (1) to introduce a new possibility of integrating matching methods and, (2) to illustrate this new method by the introduction of JSI OntoLight.

1.4 Outline of the deliverable

The present deliverable is a prototype report that describes the work that has been carried out for integrating the OntoLight system within the Alignment server and in consequence to the NeOn toolkit.

We first present the principles which govern this new kind of connection and the rational for these (Chapter 2). We then provide the specifications of the Web service to be implemented in order to offer a front-end to the Alignment server (Chapter 3). Its actual connection to OntoLight is presented (Chapter 4) before showing an example of using OntoLight through the NeOn toolkit (Chapter 5).

Chapter 2

Overview of the connection

In principle, the Alignment server works just by embedding matching methods instantiating the alignment API and make these available directly to the NeOn Alignment plug-in of the NeOn toolkit. However, it happens that some matchers cannot be easily embedded, either because they rely on local resources (databases, machines) or they are too difficult to embbed (written in a particular way).

We had to take this problem into account in order to take advantage of OntoLight from the Alignment server. We present here the solution that has been chosen and its rationale.

2.1 Principles

The retained integration principle is that the Alignment server will invoke OntoLight through a Web service interface. So the service had to be implemented on the OntoLight side and a client on the Alignment server side.

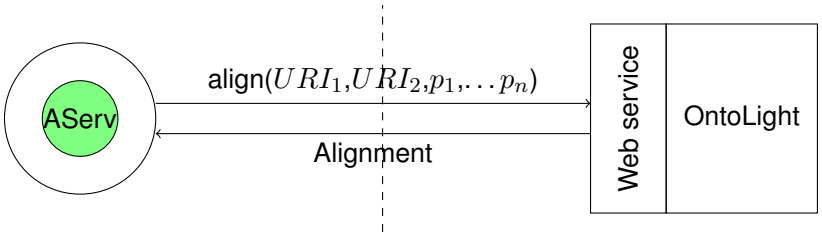

Figure 2.1: Principles of Alignment server/OntoLight connection.

The rationale of this choice is the following:

- this prevents the language mismatch problem: the Alignment server is in Java, OntoLight is in C++:
- this avoids putting the heavy computational burden on the Alignment server; OntoLight manipulates large resources that are not easily displaceable.

In addition, there are other benefits to this architecture:

- the Alignment server can use this interface to connect to other similar systems (we plan to do it with Scarlet);
- OntoLight can use this interface to be connected to other contexts.

The proposed solution only relies on standard technology (web services).

2.2 Web service extension to OntoLight

A web service for exposing OntoLight has been implemented. This allows for an easy connection with Alignment server through HTTP/SOAP protocol. The selected method of integration is beneficial twofold. First it enables an easy integration between OntoLight and the Alignment server, escaping any programming language or communication barrier and second it opens OntoLight to the general public, providing readily accessible alignment services to other possible web service consumers.
The interface of the OntoLight web service provides one method. The signature of the method is following:

```
align(url1, url2, params)
```

This method will accept two URLs as an input. They designate the physical location of the ontologies that will be aligned. The parameter "params" provides an easy implementation of additional parameters (not used in OntoLight). The output of the method is an alignment of the two ontologies, written in the RDF/XML format. A strict technical definition is given in (§4.1).

This very small interface of merely one method is in accordance to the general rule of web services trying to minimize the needed communication between modules. Further methods can be easily added as they become needed.

2.3 Webservice alignment method for the Alignment server

The required web service client has been implemented in the Alignment API as yet another matching method. In fact it is implemented as an abstract matching method so that it can be implemented diferently for different external methods.

This method, `fr.inrialpes.exmo.align.service.WSAlignment` is an extension of `URIAlignment` (in which matching items are identified by URIs).
The classical protocol of the alignment API consist of calling:

```
init( URI, URI )
```

with the URL where to find the two ontologies to match, before calling

```
align( Alignment, Parameters )
```

with the parameters of the method and an intial alignment if necessary.

In the present case, `init( URI, URI )` records the URIs and `align( Alignment, Parameters )` connects to the server identified in the `wserver` parameter, call the `align` action with the corresponding arguments and read the alignment from the returned SOAP message.

The returned alignment, in the Alignment format, is directly parsed into the current alignment.

This alignment is then visible within the alignment server and can be used and manipulated just like any other alignment.

It is also possible to refine this abstract class. For instance, we could define a `si.ijs.neon.OntoLightAlignment` which only difference is to have a fixed value for `wserver` so that the server does not have to care about it. This can be refined further by changing the whole communication interface, but this is not our purpose here.

2.4 Synthesis

A lightweight and general interface from the Alignment server and remote matchers has been implemented in the Alignment server following this model. Before testing it against OntoLight, we have tested it against the Alignment server itself (so an instance of the server can call another instance or itself).

We will now specify more precisely what is expected by the Alignment server to be connected to an external matching method.

Chapter 3

Web service specification

The principles above have been implemented in the `fr.inrialpes.exmo.align.service.WSAlignment` class. In oder to be called by the Alignment server, the external alignment method, must then provide the expected web service interface. We first provide an example of messages exchanged between the client and the web service (§3.1), before providing the WSDL specification of the web service interface to be implemented (§3.2).

3.1 Informal description

The Web service interface consists of only one action named `align` using the following parameters:

```
<url1>URI</url1>
<url2>URI</url2>
<init>URI</init>
<param name="String">String</param>
...
<param name="String">String</param>
```

`url1` and `url2` are the URL of the two ontologies to be aligned. `init` is the URI of an initial alignment to start with when this is needed (it is the first argument of the `align` method of the Alignment API. The other parameters must be identified by their names.

The result is basically an Alignment serialised with the RDF/XML format.

For being more concrete about the type of message, here are two full examples in SOAP (sent with action `align`):

```
<SOAP-ENV:Envelope xmlns='http://exmo.inrialpes.fr/align/service'
                   xml:base='http://exmo.inrialpes.fr/align/service'
                   xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
                   xmlns:xsi='http://www.w3.org/1999/XMLSchema-instance'
                   xmlns:xsd='http://www.w3.org/1999/XMLSchema'>
  <SOAP-ENV:Body>
    <url1>http://alignapi.gforge.inria.fr/tutorial/myOnto.owl</url1>
    <url2>http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl</url2>
    <param name="wsmethod">fr.inrialpes.exmo.align.impl.method.EditDistNameAlignment</param>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

for the call and

```
<SOAP-ENV:Envelope
   xmlns='http://exmo.inrialpes.fr/align/service'
   xml:base='http://exmo.inrialpes.fr/align/service'
   xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
   xmlns:xsi='http://www.w3.org/1999/XMLSchema-instance'
   xmlns:xsd='http://www.w3.org/1999/XMLSchema'>
  <SOAP-ENV:Body>
    <alignResponse>
```

```
      <result><rdf:RDF xmlns='http://knowledgeweb.semanticweb.org/heterogeneity/alignment#'
         xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
         xmlns:ns0='http://exmo.inrialpes.fr/align/service#'
         xmlns:xsd='http://www.w3.org/2001/XMLSchema#'
         xmlns:align='http://knowledgeweb.semanticweb.org/heterogeneity/alignment#'>
<Alignment rdf:about="http://aserv.inrialpes.fr:8089/alid/1219844210634/13">
            <xml>yes</xml>
            <level>0</level>
            <type>11</type>
            <method>fr.inrialpes.exmo.align.impl.method.EditDistNameAlignment</method>
            <id>http://aserv.inrialpes.fr:8089/alid/1216393091899/650</id>
            <ns0:stored>Fri Jul 18 16:58:16 CEST 2008</ns0:stored>
            <ns0:cached>Tue Aug 26 18:00:03 CEST 2008</ns0:cached>
            <ns0:ouri2>http://alignapi.gforge.inria.fr/tutorial/myOnto.owl</ns0:ouri2>
            <time>83</time>
            <ns0:ouri1>http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl</ns0:ouri1>
            <onto1>
              <Ontology rdf:about="http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl
               <location>http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl</location
              </Ontology>
            </onto1>
            <onto2>
              <Ontology rdf:about="http://alignapi.gforge.inria.fr/tutorial/myOnto.owl">
                <location>http://alignapi.gforge.inria.fr/tutorial/myOnto.owl</location>
              </Ontology>
            </onto2>
            <map>
              <Cell rdf:about="">
                <entity1 rdf:resource='http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex
                <entity2 rdf:resource='http://alignapi.gforge.inria.fr/tutorial/myOnto.owl#number'/>
                <relation>=</relation>
                <measure rdf:datatype='http://www.w3.org/2001/XMLSchema#float'>0.962962962962963</me
              </Cell>
            </map>
            <map>
              <Cell rdf:about="">
                <entity1 rdf:resource='http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex
                <entity2 rdf:resource='http://alignapi.gforge.inria.fr/tutorial/myOnto.owl#series'/>
               <relation>=</relation>
                <measure rdf:datatype='http://www.w3.org/2001/XMLSchema#float'>0.962962962962963</me
              </Cell>
          </Alignment>
        </rdf:RDF>
      </alignment>
    </alignResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

for the answer.

Currently, the `WSAlignement` implementation uses SOAP/HTTP for communicating with the web service but we plan to enhance it for using a RESTfull interface.

3.2 WSDL specification

This is a sample WSDL file for your alignment service. It is here for documentation purposes since `WSAlignment` does not use it. If you want to use it, replace the variables prefixed by `%` by their values.

```
<?xml version="1.0" encoding="UTF-8"?>

<wsdl:definitions
  targetNamespace="http://exmo.inrialpes.fr/align/service"
  xmlns:impl="http://exmo.inrialpes.fr/align/service"
  xmlns:intf="http://exmo.inrialpes.fr/align/service"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <!-- Type definitions -->

  <wsdl:types>
    <xsd:schema targetNamespace="http://exmo.inrialpes.fr/align/service">
```

```
        <xsd:element name="url1" type="xsd:string"/>
        <xsd:element name="url2" type="xsd:string"/>
        <xsd:complexType name="OptionalAlignArgs">
              <xsd:sequence>
                  <xsd:element name="init" type="xsd:string"
                                           minOccurs="0" maxOccurs="1"/>
                 <xsd:element name="params"  type="impl:Parameter"
                                           minOccurs="0" maxOccurs="unbounded"/>
              </xsd:sequence>
        </xsd:complexType>
        <xsd:complexType name="impl:Parameter">
             <xsd:simpleContent>
                <xsd:extension base="xsd:string">
                    <xsd:attribute name="name" type="xsd:string" />
                 </xsd:extension>
              </xsd:simpleContent>
        </xsd:complexType>
      </xsd:schema>
  </wsdl:types>

  <!-- Message definitions -->

   <wsdl:message name="align">
      <wsdl:part name="url1" element="impl:url1"/>
      <wsdl:part name="url2" element="impl:url2"/>
      <wsdl:part name="params" type="impl:OptionalAlignArgs"/>
   </wsdl:message>
   <wsdl:message name="alignResponse">
      <wsdl:part name="alignment" type="xsd:string"/>
   </wsdl:message>

   <!-- Port definition -->

   <wsdl:portType name="WSAlignSVC">
     <wsdl:operation name="align" parameterOrder="url1 url2">
       <wsdl:input message="impl:align" name="align"/>
       <wsdl:output message="impl:alignResponse" name="alignResponse"/>
     </wsdl:operation>
   </wsdl:portType>

   <!-- Binding definition -->

   <wsdl:binding name="SOAPWSAlignSVC" type="impl:WSAlignSVC">
     <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
     <wsdl:operation name="align">
       <wsdlsoap:operation soapAction=""/>
       <wsdl:input name="align">
         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://exmo.inrialpes.fr/align/service" use="encoded"/>
       </wsdl:input>
       <wsdl:output name="alignResponse">
         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://exmo.inrialpes.fr/align/service" use="encoded"/>
       </wsdl:output>
     </wsdl:operation>
   </wsdl:binding>

   <!-- Service declaration -->

   <wsdl:service name="WSAlignService">
     <wsdl:documentation>%MyOwnServerName%</wsdl:documentation>
     <wsdl:port binding="impl:SOAPWSAlignSVC" name="alignmentService">
       <wsdlsoap:address location="%MyOwnServerURL%"/>
     </wsdl:port>
   </wsdl:service>

</wsdl:definitions>
```

3.3 Synthesis

Once the interface used by the alignment server is defined, it is possible to implement web services providing alignments to the server. This is considered now.

Chapter 4

Connection to OntoLight

The OntoLight approach was implemented here as an alignment web service as described in the previous chapter. It can be readily consumed by the Alignment server.

The rationale for using the web service approach for OntoLight is that OntoLight uses a wealth of resources (text, thesauri, etc.) which are preprocessed locally for matching ontologies. Porting this infrastructure directly in the Alignment server would have been a burdensome task (both in terms of installing the same structure under the Alignment server and in terms of load for the Alignment server computer). In addition, the web service solution for the Alignment server is a very convenient solution which fits well in current web architecture.

In this chapter, we first give a short description of OntoLight (§4.1), followed by the complete specification for OntoLight web service, as specified and explained in §3.2. The address of the web service is: `http://kameleon.ijs.si/ontolight/ontolight.asmx`. Communication with the OntoLight web service is done over a HTTP/SOAP protocol. More details are given in §2.2. We provide the specific WSDL of the OntoLight web service in §4.2.

4.1 OntoLight Approach

OntoLight [Grobelnik *et al.*, 2008] as proposed in NeOn D3.2.1 was designed as a pragmatic approach to using large-scale ontologies as contexts. The approach is based on a light-weight ontology model and grounding of the ontology concepts in textual documents. OntoLight supports (1) ontology population where new textual instances without a known class can be classified into the selected ontology and (2) soft (probabilistic) mappings between a pair of selected ontologies can be computed, thus providing a contextual relationship between the ontologies. The later is what we are interested in here. The ontology model used in OntoLight is a relatively simple model which covers most of the well known light-weight ontologies. The model we use is a subset of richer ontology formalisms (such as OWL) in the sense that richer ontologies could be imported but not all their expressiveness can be used. Informally, the light-weight ontology model is defined by:

- A list of languages used for lexical terms.
- A list of class-types used for representing different types of nodes in the ontology structure.
- A list of classes where each class can have several lexical representations in one or several languages. One class represents one node in the graph.
- A list of relation-types used to label relations (links) between classes in the ontology graph.
- A list of relations connecting classes in the ontology graph.
- Each ontology can have one or several grounding models. Each grounding model is a function which proposes zero, one or more classes for a given instance. This corresponds to a classification/categorization model in machine learning terminology.

The OntoLight software package [NeOn D3.2.1] consists of several executable modules and a data library of ontologies. The main functionality is the contextualization of ontologies through generation of soft mappings between ontologies, thus enabling to view concepts of one ontology through the perspective of another one. OntoLight currently incorporates the following five ontologies: AgroVoc and ASFA (relevant for the Food and Agricultural Organization of the UN), EuroVoc (EU legislation), Cyc (common-sense knowledge) and DMoz (a WWW directory).

4.2 OntoLight WSDL

In this section is given the WSDL file used to consume the OntoLight web service. It is made in accordance with Alignment server requirements, but can be also used to retrieve OntoLight alignment services by the general public directly. Since OntoLight web service is running on an IIS web server, to use a custom WSDL specified in (§3.2) first a short wrapper WSDL must be auto generated. It can be found at `http://kameleon.ijs.si/ontolight/ontolight.asmx?WSDL` and is given below:

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tns="http://kameleon.ijs.si/ontolight"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
targetNamespace="http://kameleon.ijs.si/ontolight"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:import namespace="" location="custom.wsdl" />
  <wsdl:types />
  <wsdl:service name="ontolight">
    <wsdl:port name="MyBinding" binding="tns:MyBinding">
      <soap:address location="http://kameleon.ijs.si/ontolight/ontolight.asmx" />
    </wsdl:port>
    <wsdl:port name="MyBinding1" binding="tns:MyBinding">
      <soap12:address location="http://kameleon.ijs.si/ontolight/ontolight.asmx" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

The function of this WSDL is to import the required custom WSDL specified in (§3.2). This custom file is accessible at: http://kameleon.ijs.si/ontolight/custom.wsdl. Bellow it is written in full:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://exmo.inrialpes.fr/align/service"
xmlns:impl="http://exmo.inrialpes.fr/align/service"
xmlns:intf="http://exmo.inrialpes.fr/align/service"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <!-- Type definitions -->
    <wsdl:types>
        <xsd:schema targetNamespace="http://exmo.inrialpes.fr/align/service">
            <xsd:element name="url1" type="xsd:string"/>
            <xsd:element name="url2" type="xsd:string"/>
            <xsd:complexType name="OptionalAlignArgs">
                <xsd:sequence>
                    <xsd:element name="init" type="xsd:string" minOccurs="0" maxOccurs="1"/>
                    <xsd:element name="params" type="impl:Parameter" minOccurs="0"
                    maxOccurs="unbounded"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:complexType name="impl:Parameter">
                <xsd:simpleContent>
                    <xsd:extension base="xsd:string">
                        <xsd:attribute name="name" type="xsd:string" />
                    </xsd:extension>
                </xsd:simpleContent>
            </xsd:complexType>
```

```
        </xsd:schema>
    </wsdl:types>
    <!-- Message definitions -->
    <wsdl:message name="align">
        <wsdl:part name="url1" element="impl:url1"/>
        <wsdl:part name="url2" element="impl:url2"/>
        <wsdl:part name="params" type="impl:OptionalAlignArgs"/>
    </wsdl:message>
    <wsdl:message name="alignResponse">
        <wsdl:part name="alignment" type="xsd:string"/>
    </wsdl:message>
    <!-- Port definition -->
    <wsdl:portType name="WSAlignSVC">
        <wsdl:operation name="align" parameterOrder="url1 url2">
            <wsdl:input message="impl:align" name="align"/>
            <wsdl:output message="impl:alignResponse" name="alignResponse"/>
        </wsdl:operation>
    </wsdl:portType>
    <!-- Binding definition -->
    <wsdl:binding name="SOAPWSAlignSVC" type="impl:WSAlignSVC">
        <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="align">
            <wsdlsoap:operation soapAction=""/>
            <wsdl:input name="align">
                <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="http://exmo.inrialpes.fr/align/service" use="encoded"/>
            </wsdl:input>
            <wsdl:output name="alignResponse">
                <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="http://exmo.inrialpes.fr/align/service" use="encoded"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <!-- Service declaration -->
    <wsdl:service name="WSAlignService">
        <wsdl:documentation>ontolight</wsdl:documentation>
        <wsdl:port binding="impl:SOAPWSAlignSVC" name="alignmentService">
            <wsdlsoap:address location="kameleon.ijs.si/ontolight"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Chapter 5

Example

As a running example of the use of the OntoLight matcher within the Alignment server, we use the HTTP interface of the server.
The invocation, is shown in Figure 5.1.

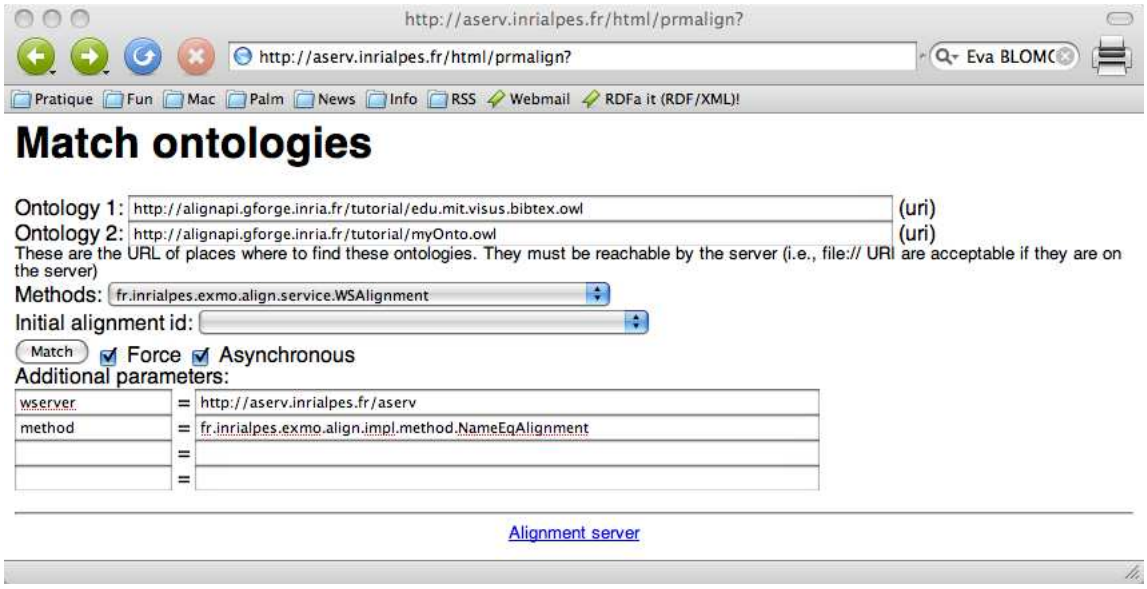

Figure 5.1: Call to OntoLight through the Alignment server. The call uses the special `WSAlignment` alignment class and passes it the `wserver` parameter. Other parameters can be passed as well.

Chapter 6

Conclusion

We have proposed a very simple and flexible interface for exposing ontology matchers to the Alignment server and thus making it available within the NeOn Alignment plug-in. It is simple because it consists of a simple operation using standard technologies (SOAP over HTTP); it is flexible because any type of parameter can be passed to the service in order to tune its behavior.

This interface has been tested against the Alignment server itself (which offers connection as alignment service) and against OntoLight. It could be used as well for connecting the server to Scarlet.

Appendix A

Installing the Alignment server

More information about the Alignment server and the NeOn Alignment plug-in are provided in D3.3.2. The developments described by this deliverable are part of the Alignment server since version 3.4. The described version is 3.5.

A.1 Get and install the Alignment server

The last version of the Alignment server can be obtained from the `http://alignapi.gforge.inria.fr` web site. Download and unzip the zip file corresponding to the desired version.

Running the Alignment server has been described in Appendix A of Deliverable D3.3.1 and in Appendix A of the Alignment API and server documentation.

We assume that the directory `$LIB` contains the jar files that come with the server release (they can be moved anywhere).

A.2 Testing your alignment service in command-line mode

In order to help developing and testing alignment web services without the burden of running an Alignment server, we provided a command line environment. It is an adaptation of the `AlignmentClient` example of the Alignment server. This is a simple stand alone java program that is meant to show how to call the Alignment server through it web service interface. The command line client is in the `examples/wservice` directory of the alignment API.

It can be compiled, if necessary, through:

```
$ javac -cp $LIB/procalign.jar AlignmentClient.java
```

For calling, the alignment service at URL `$SERV`, it is enough to do:

```
$ java -cp .:$LIB/procalign.jar AlignmentClient -S $SERV
      align
      http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl
      http://alignapi.gforge.inria.fr/tutorial/myOnto.owl
```

This will printout the returned message, containing the alignment, on the standard output. This printout is similar to the one in §3.

More output can be obtained by using the `-d` option.

A.3 Launching the alignment server

We have added in the build file for the alignment server a target `aserv` which regenerates a proxy jar file `lib/aserv.jar` listing the extension libraries that have to be embedded within the server. The server will

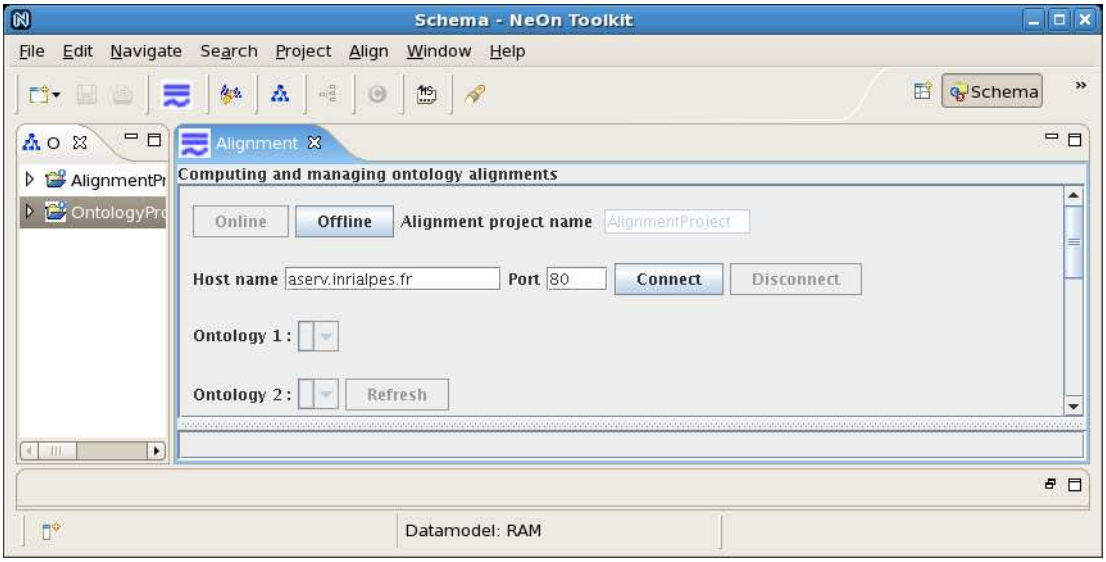

Figure A.1: Connecting to the INRIA Alignment server.

find out the embedded matchers from this list of libraries. So, it may be necessary to modify the build file (more precisely the `manifest` declaration of the `aserv` target) to embed new matchers.
The full sequence for launching the server is:

```
$ ant aserv
$ java -Djava.library.path=$WNJNLIB -jar $LIB/aserv.jar -Dwndict=$WNHOME/dict -O -H -W &
```

Bibliography

[Euzenat, 2004] Jérôme Euzenat. An API for ontology alignment. In *Proc. 3rd International Semantic Web Conference (ISWC)*, volume 3298 of *Lecture notes in computer science*, pages 698–712, Hiroshima (JP), 2004.

[Grobelnik *et al.*, 2008] Marko Grobelnik, Janez Brank, Blaž Fortuna, and Igor Mozetič. Contextualizing ontologies with ontolight: A pragmatic approach. *Journal Informatica*, 32:79–84, 2008.

[Haase *et al.*, 2006] Peter Haase, Pascal Hitzler, Sebastian Rudolph, Guilin Qi, Marko Grobelnik, Igor Mozetič, Damjan Bojadžiev, Jerome Euzenat, Mathieu d'Aquin, Aldo Gangemi, and Carola Catenacci. Context languages – state of the art. Deliverable D3.1.1, NeOn, 2006.

[Le Duc *et al.*, 2008] Chan Le Duc, Mathieu d'Aquin, Jesus Barrasa, Jérôme David, Jérôme Euzenat, Raúl Palma, Rosario Plaza, Marta Sabou, and Boris Villazón-Terrazas. Matching ontologies for context: The neon alignment plug-in. deliverable 3.3.2, NeOn, 2008.